Improving Safeguards and Functionality in Industrial Collaborative Robot HMIs through GUI Automation

Conference Paper · October 2020

CITATIONS
0

READS
14

3 authors, including:

Tudor Ionescu
TU Wien
33 PUBLICATIONS **102** CITATIONS

SEE PROFILE

Improving Safeguards and Functionality in Industrial Collaborative Robot HMIs through GUI Automation

Tudor B. Ionescu
Vienna University of Technology
Theresianumgasse 27
1040 Vienna, Austria
tudor.ionescu@tuwien.ac.at

Joachim Fröhlich
Siemens AG
Otto-Hahn-Ring 6
81739 München, Germany
froehlich.joachim@siemens.com

Markus Lachenmayr
Siemens AG
Otto-Hahn-Ring 6
81739 München, Germany
markus.lachenmayr@siemens.com

***Abstract*—Due to safety and security concerns, industrial human-machine interfaces (HMIs) cannot be updated as frequently and easily as other software. This leads to an innovation gap in productive industrial environments, which is aggravated by the demand that innovation yield immediate gains in productivity and profitability. Hence, industrial HMIs often lag behind the state of the art, which may pose risks in terms of the safety and security of human-machine interactive systems, such as collaborative robots, while impeding their interoperability with machine vision and other novel technologies. In response, this paper explores the potential and limits of novel tools and techniques from the domain of graphical user interface (GUI) automation, which can be used to extend existing robot HMIs in a flexible and pragmatic way. In this sense, the paper introduces a novel architecture for HMI automation and evaluates it using five non-exhaustive use cases for collaborative robots. Our evaluation demonstrates the effectiveness of the approach in various industrial application contexts that require specific extensions to off-the-shelf robotic software.**

Keywords—collaborative robots, HMI, GUI automation, safety, speech recognition, object recognition, patterns, templates

I. Introduction

Due to security and safety restrictions governing industrial manufacturing environments, industrial automation software, including HMIs, cannot be updated as frequently and easily as other software. Hence, HMIs used in productive industrial environments often lag behind the state of the art in terms of functionality and user experience. In such environments, this situation is aggravated by the demand that, innovation must yield immediate gains in productivity and profitability. In this context, this paper reports on an assessment of the potential and limits of the latest GUI automation tools and techniques in bridging the innovation delay affecting industrial HMIs. Modern GUI automation tools leverage computer vision and machine learning algorithms to process visual elements in GUIs in near real time and to emulate the user interactions required to react to those events. While such tools are commonly used for testing interactive software systems [1], GUI automation technologies also enables non-intrusive extension of industrial HMIs.

GUI automation has proven successful in business process automation and public administration domains [2] (in which it is better known as *robotic process automation* [3]), where vendor lock-in is commonplace (e.g., through software such as Microsoft Office and SAP). While in these domains, robotic process automation is often invoked as an alternative to outsourcing repetitive, low value-added operations [3], in the industrial domain application integrators are confronted with the heterogeneity of proprietary software systems. In response, some manufacturing companies choose to homogenize their machine fleets by using a single vendor. In this context, GUI automation can potentially help to cope with the heterogeneity of industrial automation systems by offering a means for configuring and programming them at a meta level. This paper focusses on collaborative industrial robots (cobots), which pose challenges for application integrators (i.e., manufacturing companies using them for productive purposes) in terms of safety, security, flexibility, adaptability and interoperability. Nevertheless, the proposed approach is applicable to any automation system that can be operated and/or programmed via its proprietary or non-proprietary HMI.

The paper begins with a description of these challenges and a literature review before introducing three usage contexts for HMI automation—a factory, a small manufacturing company, and a makerspace (i.e., a shared machine shop, which is open to the public). These scenarios are arguably representative of current cobot usage contexts in smart manufacturing. We then introduce a novel architecture for enabling HMI automation for industrial robots using existing GUI automation tools and techniques and describe five use cases that were implemented and tested in an experimental industrial environment. Then we relate and evaluate the use cases to the three usage contexts while discussing the limitations of the approach.

II. Background: Challenges to Cobot Adoption

Cobots can work near humans without a safety fence and thus enable more flexible human-robot interactions in assembly automation. Yet the vision of hybrid automation, in which humans and robots work together in a symbiotic way [4], is challenged by safety and flexibility issues. Cobot applications involving direct interaction with humans require safety certifications based on thorough risk analyses [5]. This requires manufacturers to conduct a new risk assessment after changing the layout and program of an application, which introduces additional costs. Another challenge consists in the limitations on how cobots can be programmed and operated. The programming environments embedded in HMIs, which run on teaching pendants, often build on proprietary programming models and

languages. Extending these environments is limited to the possibilities offered by vendor-provided APIs, which foster hardware-based interoperability with other systems and modules (i.e., using physical ports and protocols such as MODBUS or OPC UA) over software-based integration (i.e., using modular software extensions and/or cloud integration). Other vendors provide high-level language support in offline programming environments that allow users to download programs written in higher-level, non-proprietary languages like Java or C++ to the robot's control computer. Yet this method is usually reserved for robotics experts and professional software engineers—a scarce human resource in the industrial automation domain. Today cobots can also be ordered, installed and operated by anyone willing to invest around 10,000 euros in a practical universal helper. Such cobots come with downloadable apps and simplified end user programming interfaces. Unfortunately, the cobot market lacks consistency and predictability, which makes the adoption of cobots in manufacturing and elsewhere a financially risky endeavor. Mitigating this risk arguably requires simple, pragmatic and inexpensive software solutions to reduce the overall cost of productive cobot operation, while fostering open innovation in makerspaces and elsewhere.

III. State of the Art

Currently, there exist two dominant approaches to extending the functionality of cobots: vendor-supported HMI plugins and robotic middleware. Vendors such as Franka Emika and Universal Robots (UR) provide plugins in app stores for extending the functionality of robot HMIs. However, the business models of these robot vendors build on selling costly, hardware-specific extensions or apps. Such restrictive, business-oriented strategies are an important impediment to the wider adoption of cobots by industry, especially small manufacturing companies. By contrast, most cobot vendors provide open low-level APIs that can be used to develop ROS (Robot Operating System) [6] drivers. ROS is a popular open source robotic middleware that is supported by many robot vendors and users who regularly contribute new packages and drivers. ROS provides a generalized robot programming framework, with applications that are easily portable from one robot to another. The robotic middleware approach has proven useful in research and experimental development contexts, where software need not reach a high maturity level. Yet, in an agile, result-oriented industrial environment, the advantages of ROS do not hold in all circumstances. Generic and reusable components, such as ROS drivers, are generally developed with autonomous robotic systems in mind, in which a combination of teaching and offline programming is rarely supported. Using offline and remote programming sacrifices some of the important programming capabilities and features available to users through the robots' HMI (notably safety, security, reliability, and teach-in). In addition, robot skills [7] emulating manual assembly operations cannot cover all variants of any assembly operation (e.g., screwing, transportation) [5,8] and may thus prove unfeasible in practice. This situation reveals a technology gap with respect to extending existing robot HMIs in an agile and pragmatic way.

Concerning the use of GUI automation techniques in the industrial automation domain, Huesman [18] used such techniques for testing NASA's Spaceport Command and Control System. Kasper et al. [9] introduce a method for abstracting specific robotic perception and control functions by embedding them in a GUI automation scripting language for the purpose of simplifying robot programming. Using Sikuli (a GUI automation tool), the authors perform on-screen visual pattern recognition of objects visualized in a ROS-based robot simulation environment. The robot system reacts upon detection of objects in the simulation by executing certain courses of action. Polden et al. [10] employ a GUI automation tool to manipulate the user interface of a welding robot for testing and system interoperability purposes. GUI automation tools have also been used for automatic configuration [11], anomaly detection [12], and intrusion detection [13,14] in industrial control systems. Ionescu & Schlund [5] introduce a technique for extending web-based cobot programming environments using meta-programming techniques similar to GUI automation.

IV. HMI Automation: Contexts and Scenarios

A. Scenario 1: The Lot Size 1 Factory

The first usage scenario is situated in the context of a factory adopting some of the Industry 4.0 principles (e.g., lot size 1, the product steers its production process, etc.). The factory also uses human-robot collaboration applications that need to be adapted frequently to new product designs without compromising safety and security. The factory also has a training center for robot operators and uses different safety profiles in the training center and on the shop floor in accordance with the risk assessments conducted by external safety consultants. As new operators complete their training, they must first gain experience with using the robot for productive purposes, which requires a gradual increase of the robot's force and speed depending on user experience. After sufficient practice hours, users are allowed to operate the robot at its full potential. Some robot applications are integrated into assembly lines that require some degree of interoperability between the robots and other production units on the assembly line. In a lot size 1 scenario, in which two consecutive product units require different assembly operations to be performed, the robots need to communicate with other systems (e.g., a camera) in order to determine which production steps need to be performed on the product being detected by the camera. The factory uses both collaborative and non-collaborative robot applications. For the latter, the human workers intervene in cases in which errors occur and the robot state needs to recover.

B. Scenario 2: The Small Manufacturing Company

As opposed to hierarchically managed factories employing numerous workers and managers, the small manufacturing company (SMC) has more flexibility in organizing production and in conducting human-robot interaction safety. Current cobot safety standards, such as ISO/DIS 10218 and ISO/TS 15066, require application integrators to conduct case-by-case risk assessments. This includes physical force measurements that are contingent on the layout of the assembly station and the application program. A risk assessment conducted by an external safety consultant provides organizations with the legal protection in case of a work accident involving a cobot application. Yet whenever any of the features of a cobot application changes, a new risk assessment is required. Case-by-case cobot risk assessments thus reduce the flexibility and

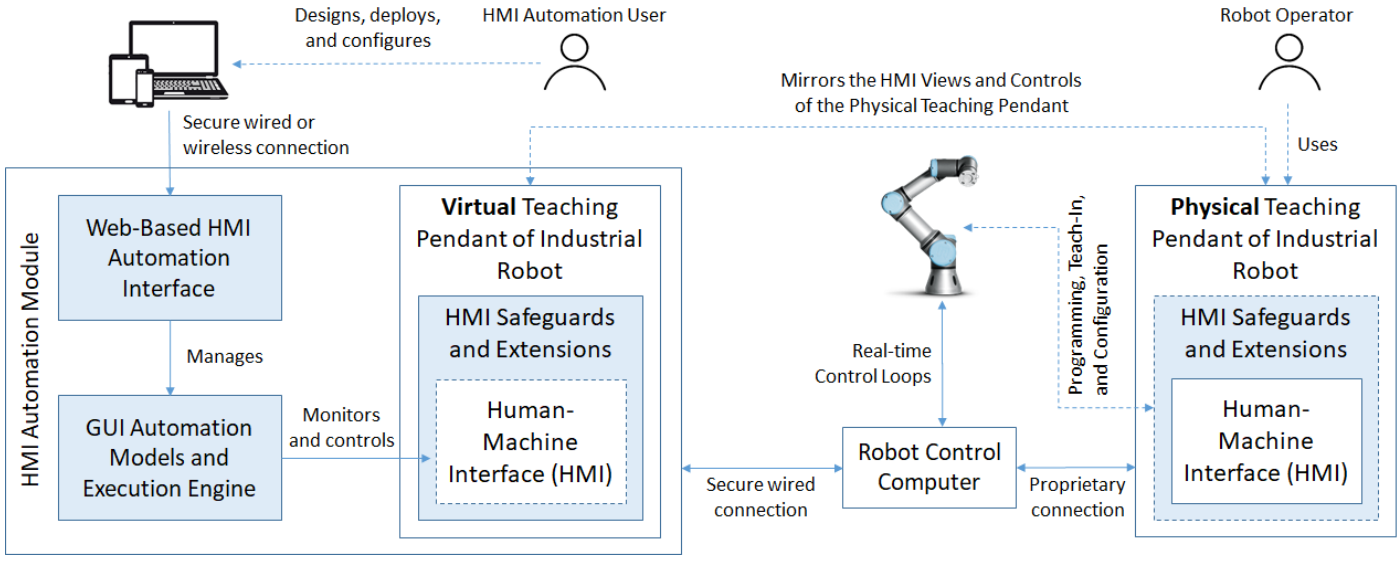

Fig. 1. Architecture of cobot HMI automation. White blocks represent components available from robot vendors or third-party providers. Blue blocks represent components and enhancements required for enabling HMI automation.

versatility of cobot applications while increasing their overall cost, especially in small batch production. Hence, the SMC is more interested in investing in comprehensive employee training and allowing staff to operate and program the cobots directly, as this is more cost-effective than conducting case-by-case risk assessments, provided that power and force limitations are in effect all the time. While these limitations reduce the speed and productivity of cobots, they do not impede their flexibility. In SMCs employing a small number of well-trained employees, cobots can thus be used in a flexible, more agile way for small batch production. Simultaneously, the SMC is less compelled to invest in novel but complex technical solutions when simpler, more pragmatic solutions are available.

C. *Scenario 3: The Shared Machine Shop*

Shared machine shops or makerspaces emerged in the past two decades as places where diverse people having no specific institutional affiliation can lease use manufacturing equipment for their own purposes. While 3D printers, laser cutters, and other additive and subtractive manufacturing technologies have dominated the makerspace landscape for years, today there is a growing interest in industrial robots and cobots (see, e.g., [15]). Makerspaces are interesting not only for cobot vendors who increasingly target non-industrial markets, but also for robotics researchers who seek to simplify certification procedures in the industry and to leverage the open innovation potential of shared machine shops. Makerspaces differ from factories and SMCs in that user applications are not known in advance because their members can use the equipment at their discretion. Makerspace members are a category of cobot users who seek to accomplish various tasks for useful or creative purposes without necessarily undergoing specialized training. To minimize their own liability in the case of accidents while maintaining the attractivity of robots as shared machines, makerspaces must find innovative ways to ensure the safety of human-robot interactions while making robots more versatile and their programming and operation easier.

V. HMI Automation: Architecture and Use Cases

Fig. 1 illustrates the architecture of our proposed approach when applied to a typical robotic system composed of a robot arm, a control computer, and a teach pendant hosting the HMI. To enable robot HMI automation, the HMI software hosted on the physical teaching pendant is mirrored using a secure wired connection (e.g., USB, Ethernet) and a remote GUI access tool running on the HMI automation module (i.e., an industrial computer or edge device). For example, Universal Robots allows mirroring the robot's HMI using VNC [16]. Other vendors (e.g., KUKA, ABB, Fanuc) offer similar open or proprietary solutions for remotely monitoring and controlling a robot's HMI in operation. The industrial computer (or edge device) hosts a desktop GUI automation engine, such as Sikuli [17] or SikuliX [19], which is managed by the user via a web-based interface. Using this interface, the user designs, deploys and configures GUI automation models (e.g., scripts or workflows) that are executed by the GUI automation engine. As a result, while the robot operator uses only the physical teaching pendant of the robot, the GUI automation engine monitors and manipulates the same HMI via the virtual teaching pendant. The HMI automation module thus acts as a second operator that assists the human operator in both routine and specialized scenarios (exemplified in the next sections).

The HMI automation model allows users to define custom HMI monitors that trigger an event whenever a certain condition is detected (e.g., certain GUI elements—buttons, sliders, labels, etc.—appear or disappear). Depending on the detected conditions, the HMI automation module can perform the same actions upon the HMI as the operator, e.g., clicking and tapping, typing text, dragging and dropping GUI elements. These features are facilitated by graphical processing libraries such as OpenCV, which enable the recognition of certain graphical patterns on a computer screen in near real time. To emulate user actions in reaction to detection of certain conditions, the HMI automation module manipulates the GUI elements of the HMI

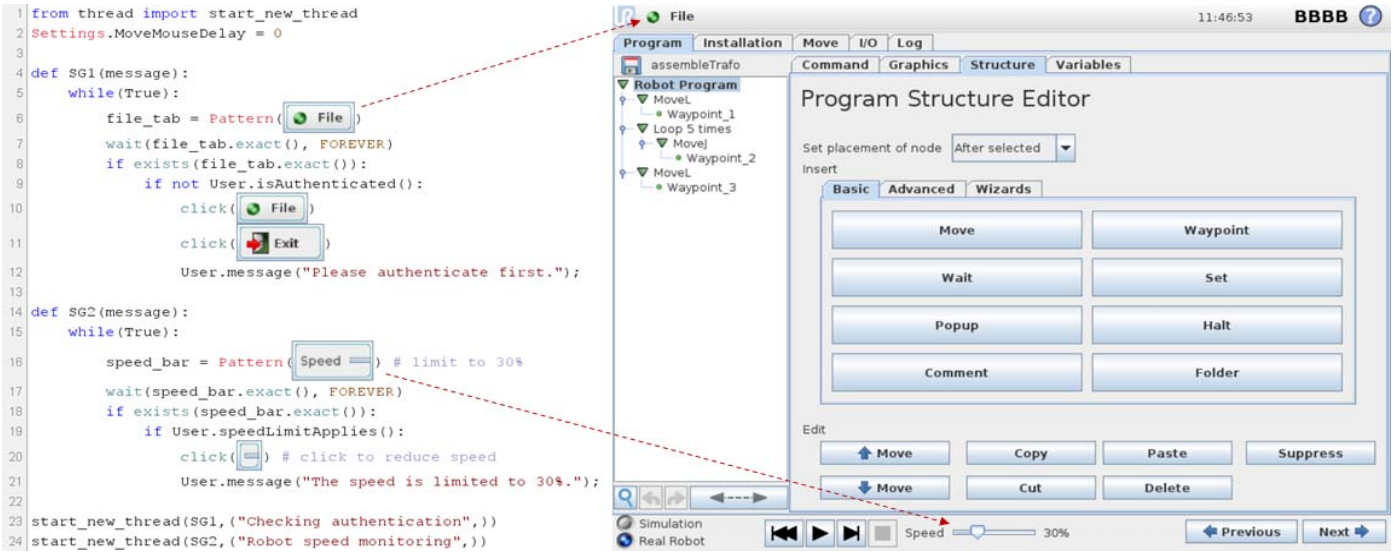

Fig. 2. Implementation of authentication and authorization safeguards that monitor and interact with Polyscope—the HMI of the UR5 CB series robot (UC 1).

using the same means as the human operator. This is facilitated by GUI automation tools that are commonly used for testing graphical user interfaces.

In the following, we introduce five non-exhaustive use cases for cobot HMI automation related to the scenarios described above. The use cases were selected in consideration of the challenges to cobot adoption; notably safety, multi-modal programming and hardware/software interoperability issues. To evaluate these use cases, we implemented a series of safeguards and functional extensions to the HMI of a UR5 robot—called Polyscope; using SikuliX [19]—a popular open-source GUI automation tool uses Python for scripting and offers means for directly capturing visual patterns of interests as screenshots (e.g., different input fields or any other on-screen image). Polyscope is a typical cobot end-user programming environment that uses a combination of textual and graphical programming elements and includes a basic robot simulator. More details about SikuliX and Polyscope are provided as needed as part of the use case descriptions below.

We chose to evaluate HMI automation using Polyscope because Universal Robots is one of the most popular industrial-grade cobot vendors used by many manufacturing companies, both large and small, and some makerspaces around the world. Accordingly, their Polyscope HMI is also very popular [20]. Universal Robots also provides an open and flexible API which can be used to create custom applications; however, this does not allow extending or modifying the core features of Polyscope. In this context, HMI automation complements the capabilities of the official API by allowing users to extend and modify the behavior of the vendor-provided HMI software (Polyscope). This arguably provides a sound basis for comparing different approaches to developing, extending and modifying robotic software—i.e., using vendor APIs vs. HMI automation.

A. UC 1: Safety and Security Safeguards

The UR5 cobot offers a basic, single-user authentication and authorization model based on two passwords—one for the general usage of the robot and one for the safety settings. This use case focuses on providing an additional safeguard to UR5's HMI that extends its security features by enabling role-based authentication and authorization. In order to operate the robot, the user must first authenticate, e.g., by inserting a smart card into a smart card reader, which sets the current user's identity in the configuration of the HMI automation module. This module executes the authentication (SG1) and authorization (SG2) safeguards shown in Fig. 2. The two safeguards run in separate threads and monitor the *File* menu and speed bar graphical elements in the robot's HMI. The *File* menu is visible only in the views for running and creating robot programs, which can be accessed only by authenticated users. SG1 thus asserts that the current user is authenticated; otherwise it simply exits the programming view by clicking on the *File* menu and subsequently on the *Exit* menu item within a fraction of a second. The user is informed about the reason for this program behavior using the means implemented by the HMI automation module (e.g., visual or audio messages). If the user is properly authenticated, SG2 monitors the speed bar at the bottom of the screen to detect whether the user attempts to increase the speed beyond the limit imposed by specific roles. In line with the scenarios described above, when a trainee or an inexperienced user operates the robot, the SG2 safeguard will automatically reduce the robot speed to a predefined upper limit (30% in this example). To accomplish this, SG2 monitors the speed bar and clicks on its lower range whenever the speed is set to a value beyond the limit associated with the user's role. The HMI automation module thus extends the safety and security features of Polyscope in a non-intrusive way.

B. UC 2: Ergonomic Teach-In Using Voice Commands

The UR5 robot HMI provides a Freedrive button, which – when kept pressed – enables the user to manually drag the robot to a desired position to the end of teaching a waypoint. Yet keeping the Freedrive button pressed with one hand for longer periods, while manipulating the robot using the other hand is cumbersome and induces physical strains on the wrist. More ergonomic hardware-based solutions include a mountable flange ring with a button that activates the robot's Freedrive mode [21]. This solution, however, adds around 1500€ to the cost of the robot system. Using HMI automation, a simpler and inexpensive

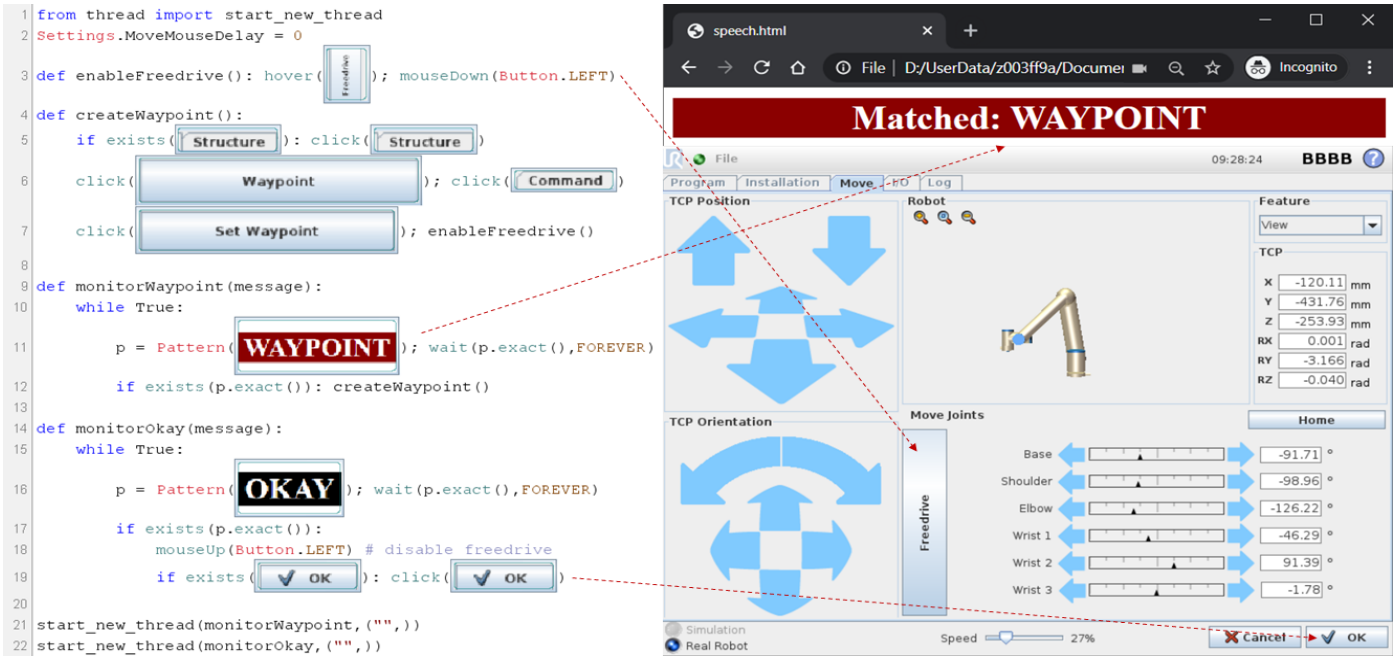

Fig. 3. SikuliX implementation (Python) of ergonomic teach-in using voice commands for the UR5 robot (UC 2).

alternative can be implemented using speech-based commands. To demonstrate this, we used the W3C web-speech API [23] to recognize two commands: The "Waypoint" command adds a new waypoint to the current program by emulating the necessary HMI user interactions and then activates Freedrive mode. The "Okay" voice command deactivates Freedrive mode and saves the coordinates of the waypoint.

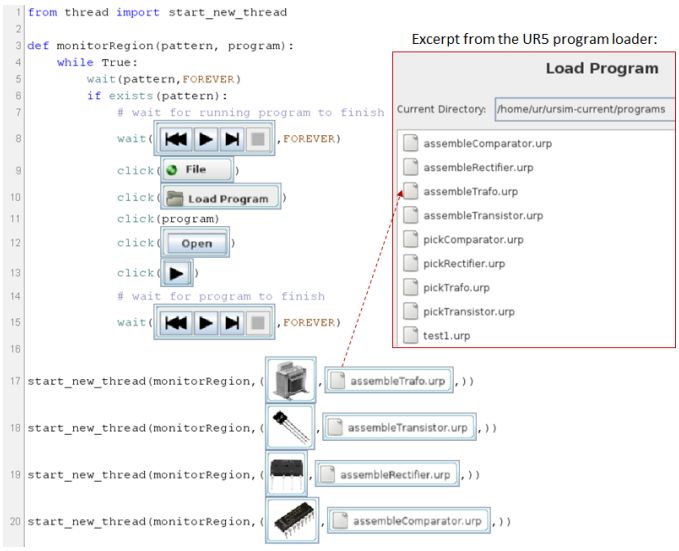

Fig. 4. HMI automation model monitoring four regions of a multiple display and executing a program associated with the recognition of a certain pattern.

Figure 3 illustrates the implementation of this use case. A simple speech-enabled web page is kept open in a browser window on top of the UR5 HMI. The web speech API is configured to match the two commands and to display a large banner with the matched command for three seconds. The HMI automation program monitors these commands in separate threads and performs the necessary user actions for creating a waypoint, activating Freedrive mode by keeping the corresponding button pressed, and for deactivating Freedrive and storing the waypoint. When the Freedrive mode is activated, the user can move the robot to the desired position. This position is stored when the "Okay" command is issued. Hence, the user can store multiple waypoints by repeating a simple three-step workflow, which saves about 50% of the time needed to accomplish the same operations manually. Note that in this case, the HMI automation engine concomitantly monitors two different GUIs (i.e., the browser window and Polyscope) and manipulates the latter using only 22 lines of code. Additional voice commands can be implemented in a similar, e.g., by speeding up robot programming and enabling people with various forms of impairment to operate robots.

C. UC 3: Robot Control Using Visual Pattern Recognition

Using similar mechanisms as in the previous use case, HMI automation enables the control of the robot upon recognition of certain visual patterns on screen. These patterns could be in the form of pictures of workpieces captured by a camera. When creating the HMI automation script, screenshots of the exact images as captured by the camera can be used as the visual patterns being monitored. Upon recognition of a pattern, the HMI automation module can load and execute a robot program in the robot's HMI corresponding to the recognized component, as shown in Fig. 4. To this end, an image preprocessor is required in order to correct eventual distortions in the captured image (i.e., scaling, rotation and translation). This can be achieved using an image registration procedure [22] supported by generic image processing libraries, such as OpenCV or Matlab's Image Processing Toolset.

In the example from Fig. 4, it is assumed that a camera captures a picture at regular intervals and displays it on a computer screen, which is not shown in Fig. 4 for space reasons. The provisioning of the four different workpieces (e.g., by another robot or a conveyor) is being monitored by the HMI

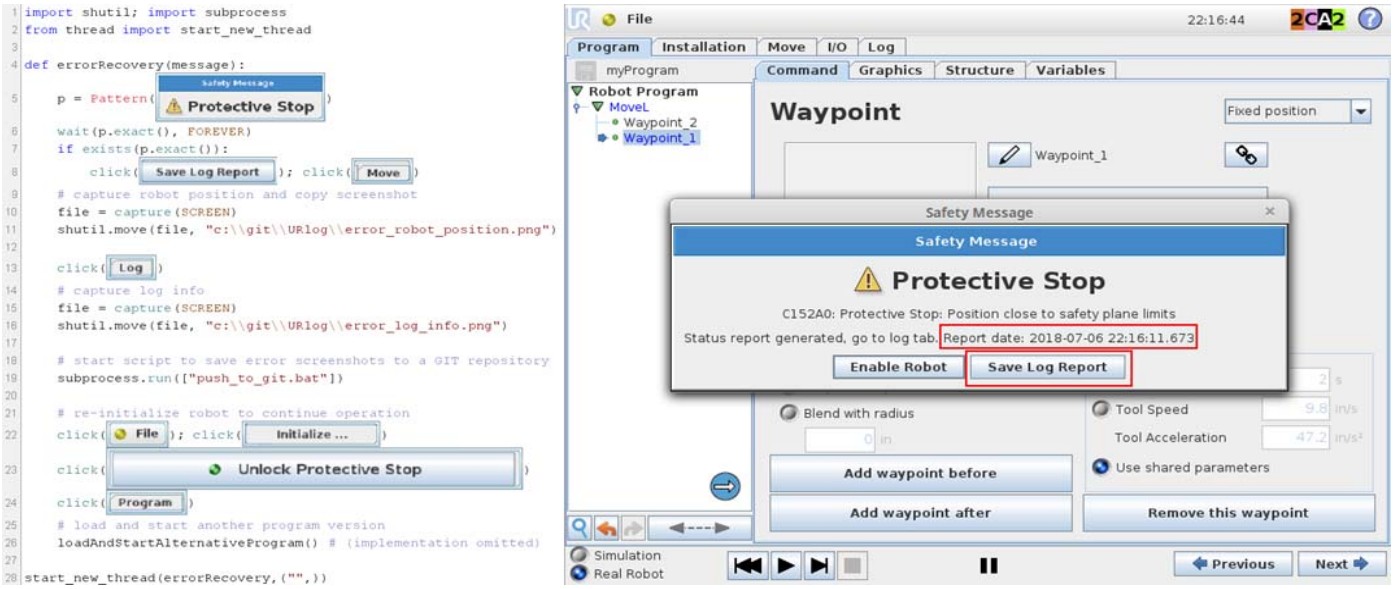

Fig. 5. HMI automation procedure to recover from a protective stop (UC 4).

automation module in different threads. When any of the components is placed in the capturing range of the camera, the image registration algorithm first corrects its alignment and orientation and then displays a greyscale version of the image on the computer screen. When any of the expected components is recognized by the thread monitoring it, the robot program corresponding to the recognized workpiece is loaded and executed. In this simple implementation, only one component can be placed in the capturing range of the camera at any given time in order to avoid triggering different programs at the same time. A more reliable solution can be obtained by implementing mutual exclusion logic between the different threads. While this simple approach assumes that the workpieces are placed in a fixed position, with the image registration step correcting only slight distortions, it also facilitates the interoperation of different hardware/software systems (i.e., the camera and the robot) without the need of a direct communication interface between them. This solution thus avoids the use of specialized hardware protocols, such as OPC UA or ModBus, as well as the development of possibly complex software modules for visual workpiece recognition. One of the benefits of this approach is that users with little programming experience can add new visual patterns and associate them with robot programs using a small number of relatively simple HMI automation commands.

D. UC 4: Error Recovery and Interoperability between HMIs

Proprietary end-user programming environments provide limited error handling and recovery capabilities. For example, if a collaborative robot stops due to a collision or some other unexpected event, human intervention is needed. In the industrial domain, however, there are situations in which collaborative robots are operated autonomously (i.e., behind safety fences or other protective panels) or semi-autonomously (i.e., in human-robot coexistence configurations, where humans are out of the reach of the robot). In these cases, an error caused by an unexpected situation, such as a misplaced workpiece leading to a collision can stop production. When entering an error state, for example, some industrial robots may need to be restarted, while most of them require a series of clicks in the robot's HMI to resume operation. These manual actions can be automated using an HMI automation procedure implementing and error recovery pattern, such as loading an alternative program and retrying.

Fig. 5 shows a protective stop error in Polyscope and the associated error recovery procedure. Upon encountering such an error, the error recovery thread detects the protective stop error message and automatically captures screenshots of different HMI views that reflect the robot's position and error context. Then, the HMI automation script first pushes the screenshots to a file versioning system (e.g. Git), then reenables the robot, loads an alternative program, and continues operation until a human can intervene to (remotely) analyze the root cause of the error in the original program. This way, production need not stop upon encountering an error. Note that, in close human-robot collaboration scenarios, collision errors should never be handled automatically since they can occur upon colliding with different parts of the human body. In such a case, automatically resuming operation could cause even more harm.

E. UC 5: Extending the Program Template Library

Robot HMIs usually offer a limited set of program patterns or templates that help to structure common assembly tasks, such as pick-and-place, drilling or palleting. The user is guided through parameterizing these program patterns using wizards. HMI automation can be used to add pattern templates by emulating a series of "boilerplate" program steps that would otherwise take considerable time to input manually using the robot's native HMI. Program pattern templates can be generated, e.g., using the web-based configuration interface of the HMI automation module. To support this feature, the HMI automation module must provide a set of reusable generator functions that can be used to create new patterns and wizards. In addition, to automate repetitive tasks, waypoints can be generated from predefined lists. This considerably speeds up programming and testing of robot applications. Fig. 6 shows an example of a pattern generator function that creates a series of waypoints required to place objects into a tray. The

configuration interface of the HMI module can be used to configure this program generation function, e.g., by providing users with parameter input forms. The generator function from Fig. 6 uses a nested loop structure to generate the waypoints corresponding to the compartments of the tray.

This use case shows how code for a proprietary robot programming environment can be generated through GUI automation using non-proprietary technologies. For Universal Robots, code generation is supported only by commercial third-party tools, such as RoboDK [24]. HMI automation thus enables developers who do not have access to proprietary source code to generate programs for the robot's vendor-provided HMI. This approach is useful especially in the context of safety certifications, since safety consultants can conduct risk assessments focused on programs developed for the robot's vendor-provided HMI rather than for a custom programming environment with which consultants are not familiar.

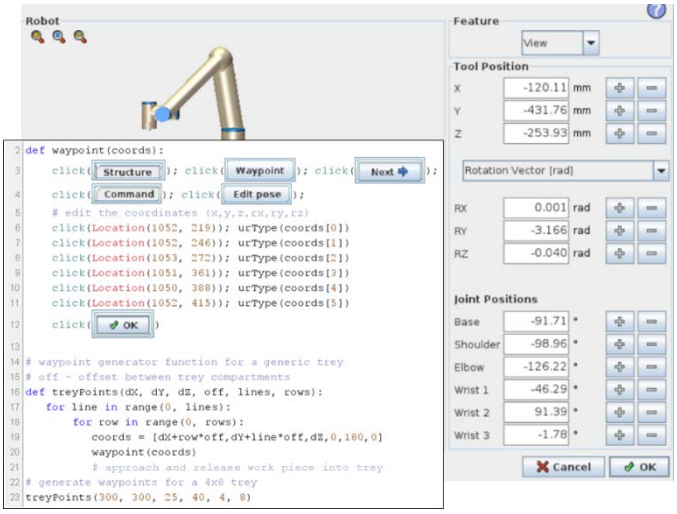

Fig. 6. *Background:* Waypoint editing screen in the UR5 HMI. *Foreground:* SikuliX waypoint and tray pattern generator functions. In this example, the script uses the hardcoded coordinates of the X, Y, Z, RX, RY, RZ input fields as an alternative to visual pattern-based localization of these fields (UC 5).

VI. Discussion

The implementations of the five use cases for evaluating HMI automation have shown that the proposed approach provides pragmatic, flexible and adaptable means for improving the safeguards and functionality of collaborative industrial robots. This helps to reduce the innovation gap in manufacturing companies and fosters the wider adoption of a versatile digital manufacturing technology. Rather than providing a one-size-fits-all solution, HMI automation may be regarded as an architectural approach for extending and adapting existing industrial automation systems and can be used in a context-dependent way. In this sense, the degree to which HMI automation can benefit different kinds of users depends on the type of organization in which it is used and on the various goals of that organization with respect to human-robot collaboration. These considerations influenced the selection of use cases introduced in this paper.

Factories and makerspaces are more likely to benefit from UC1 (safeguards) than SMCs because there is a higher diversity of individual users who operate robots than in an SMC. UC1 also provides makerspaces with a way to secure access of their members to robots by integrating robot systems with existing authentication and authorization systems such as Fabman [25]. These systems are usually correlated with the level of experience of the current user. UC1 is currently being further evaluated as part of the "Cobot Meets Makerspace" project [26].

Voice-based programming and control of robots (UC2) has been around for at least 20 years; however, it has not yet been widely adopted because of the relatively high speech recognition error rates. Today this error rate is below 5%, which makes this technology more interesting for users who prefer the teach-in technique over other programming techniques. This is likely to be the case for SMCs and makerspaces, with factories still preferring traditional graphical or textual programming techniques, also due to noise level in operational factories. Currently, there are no off-the-shelf voice control modules in Polyscope and most other commercial cobots.

Automated error recovery (UC4) is a desirable assembly system property and a current research topic. Therefore, factories and SMCs are likely to benefit from simple but effective error recovery techniques that can reduce downtimes in both small and large batch production scenarios. By contrast, in makerspaces users tend to explore and experiment rather than engage in batch production of goods. Nevertheless, as the COVID-19 crisis has shown, makerspaces can also turn into SMCs when there is a demand for specialized, easy-to-produce goods such as 3D-printed respiratory masks and ventilator components. Robots can be used for 3D-printer tending overnight, which makes error recovery interesting in autonomous or semi-autonomous production scenario. Currently, robot vendors offer remote access solutions to recover from errors during operation when human intervention on the shop floor is not possible. Remote support, however, is not free of charge. HMI automation thus provides an inexpensive alternative for various cobot application users.

The interoperability of machine vision systems and robots facilitated by UC3 (visual pattern recognition) is more likely to benefit SMCs who cannot afford expensive machine vision systems and makerspaces, where object detection applications are popular. In factories, professional camera systems are usually preferred because of their superior reliability and interoperability with industrial standards such as OPC UA and ModBus. Nevertheless, as machine vision systems become more modular and their recognition capabilities more powerful, labeling the recognized objects and monitoring these labels using an HMI automation module may render UC3 more interesting for factories as well.

Finally, extending existing vendor-provided program template collections (UC5) is a useful feature in both high and low volume production scenarios. The mechanisms described in UC5 provide a viable alternative to acquiring expensive, specialized, skill-oriented robotic software packages. Especially for SMCs and factories, UC5 fosters the development of practical, long-term know-how in the company. Robotic assembly patterns based on common program functions and structures can also be more easily shared on the Internet than robot-specific software skills.

VII. Conclusion

This paper presents an exploratory evaluation of HMI automation by the example of collaborative industrial robots. HMI automation uses techniques and tools from the domain of GUI automation. HMI automation can reduce the innovation gap, which can be observed in productive manufacturing environments by providing a pragmatic yet effective and flexible means for extending and adapting the functionality of existing robot HMIs. To explore the potential and limits of HMI automation, we introduced a novel HMI automation module architecture and exemplary implementation for a UR5 robot in five non-exhaustive use cases. This architecture has the potential to benefit mainly three types of organizations: factories, small manufacturing companies, and public makerspaces.

Our evaluation suggests that the utility of different HMI automation use cases depends on the type of organization employing them, with small manufacturing companies and makerspaces being more likely to benefit from them than factories. This evaluation also showed that current open source GUI automation tools are not completely reliable due to a relatively low yet significant visual pattern recognition error, which may sometimes lead to the misinterpretation of the visual state of the HMI being automated. However, this error rate is not due to users of graphical processing algorithms but rather to the implementation of GUI automation tools (in this case SikuliX); these become less reliable when the computer screen contains dynamic graphical elements. This effect can be mitigated by reducing the screen region being monitored and focusing on the graphical elements of interest.

In order to make industrial HMI automation more reliable and versatile, the visual pattern recognition algorithms implemented in existing GUI automation tools must be augmented by confidence indicators and failure handling mechanisms that can provide error handling in the case of erroneous interpretations of visual patterns on screen. As part of our future work, we plan to extend existing GUI automation tools by error handling features and to define and evaluate further non-functional requirements that will facilitate the future development of reliable and versatile HMI automation tools for the industrial domain.

Acknowledgment

This research was partly supported by the Austrian Research Promotion Agency (FFG) under Grant number 871459 (https://www.comemak.at).

References

[1] Nguyen, B. N., Robbins, B., Banerjee, I., & Memon, A. (2014). GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 21(1), 65-105.

[2] Yang, X., Miao, Y., & Zhang, Y. (2011). Model-driven GUI automation for efficient information exchange between heterogeneous electronic medical record systems. In Information Systems Development (pp. 799-810). Springer, New York, NY.

[3] Asatiani, A., & Penttinen, E. (2016). Turning robotic process automation into commercial success-case OpusCapita. Journal of Information Technology Teaching Cases, 6(2), 67-74.

[4] Wang, L., Gao, R., Váncza, J., Krüger, J., Wang, X. V., Makris, S., & Chryssolouris, G. (2019). Symbiotic human-robot collaborative assembly. CIRP annals, 68(2), 701-726.

[5] Ionescu, T. B., & Schlund, S. (2019). A participatory programming model for democratizing cobot technology in public and industrial Fablabs. Procedia CIRP, 81, 93-98.

[6] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009). ROS: an open-source robot operating system. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[7] Pedersen, M. R., Nalpantidis, L., Andersen, R. S., Schou, C., Bøgh, S., Krüger, V., & Madsen, O. (2016). Robot skills for manufacturing: From concept to industrial deployment. Robotics and Computer-Integrated Manufacturing, 37, 282-291.

[8] Ionescu, T. B. (2019). Developing software for the shop-floor on the shop-floor. In Proceeding of IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE.

[9] Kasper, M., Correll, N., & Yeh, T. (2014, April). Abstracting perception and manipulation in end-user robot programming using Sikuli. In 2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA) (pp. 1-6). IEEE.

[10] Polden, J., Pan, Z., Larkin, N., Van Duin, S., & Norrish, J. (2011). Offline programming for a complex welding system using DELMIA automation. In Robotic Welding, Intelligence and Automation (pp. 341-349). Springer, Berlin, Heidelberg.

[11] Warner, P. C. (2015). Automatic configuration of programmable logic controller emulators (No. AFIT-ENG-MS-15-M-024). Air Force Institute of Technology, Wright-Patterson Graduate School of Engineering, School of Engineering and Management.

[12] Corvin, C. M. (2015). A Feasibility Study on the Application of the ScriptGenE Framework as an Anomaly Detection System in Industrial Control Systems.

[13] Gallenstein, J. K. (2017). Integration of the Network and Application Layers of Automatically-Configured Programmable Logic Controller Honeypots (No. AFIT-ENG-MS-17-M-029). Air Force Institute of Technology Wright-Patterson AFB United States.

[14] Girtz, K., Mullins, B., Rice, M., & Lopez, J. (2016). Practical application layer emulation in industrial control system honeypots. In International Conference on Critical Infrastructure Protection (pp. 83-98). Springer.

[15] Cobot Meets Makerspace (25.04.2020): https://www.comemak.at

[16] Universal Robot Remote Access (25.04.2020): https://blog.universal-robots.com/industry-40-is-here

[17] Yeh, T., Chang, T. H., & Miller, R. C. (2009). Sikuli: using GUI screenshots for search and automation. In Proce. of the 22nd annual ACM Symposium on User Interface Software and Technology (pp. 183-192).

[18] Huesman, J. (2016). Spaceport Command and Control System User Interface Testing.

[19] SikuliX Website (accessed 25.04.2020): http://sikulix.com/

[20] Weintrop, D., Afzal, A., Salac, J., Francis, P., Li, B., Shepherd, D. C., & Franklin, D. (2018). Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In Proc. of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-12).

[21] Youring for Universal Robots (25.04.2020): https://www.universal-robots.com/plus/urplus-components/accessories/youring/

[22] Zitova, B., & Flusser, J. (2003). Image registration methods: a survey. Image and Vision Computing, 21(11), 977-1000.

[23] Shires, G., & Wennborg, H. (2012). Web speech API specification. Final Report, W3C.

[24] RoboDK website: https://robodk.com/

[25] Fabman website: https://fabman.io/

[26] Cobot Meets Makerspace Project website: https://www.comemak.at